Tight Time-Space Tradeoff for Mutual Exclusion

Nikhil Bansal[*] Vibhor Bhatt[†] Prasad Jayanti[‡] Ranganath Kondapally[§]

ABSTRACT

Mutual Exclusion is a fundamental problem in distributed computing, and the problem of proving upper and lower bounds on the RMR complexity of this problem has been extensively studied. Here, we give matching lower and upper bounds on how RMR complexity trades off with space. Two implications of our results are that constant RMR complexity is impossible with subpolynomial space and subpolynomial RMR complexity is impossible with constant space for cache-coherent multiprocessors, regardless of how strong the hardware synchronization operations are.

To prove these results we show that the complexity of mutual exclusion, which can be "messy" to analyze because of system details such as asynchrony and cache coherence, is captured precisely by a simple and purely combinatorial game that we design. We then derive lower and upper bounds for this game, thereby obtaining corresponding bounds for mutual exclusion. The lower bounds for the game are proved using potential functions.

Categories and Subject Descriptors

D.1.3 [**Software**]: Programming Techniques—*Concurrent programming*

Keywords

Mutual exclusion, remote memory reference (RMR), process synchronization, lower bounds, bin-pebble game, time-space tradeoff, shared memory algorithms, cache coherence.

[*]Eindhoven University of Technology, Eindhoven, the Netherlands. Email: n.bansal@tue.nl

[†]Microsoft Corporation, Redmond, WA. Email: vibhorb@microsoft.com

[‡]Department of Computer Science, Dartmouth College, NH. Email: prasad@cs.dartmouth.edu

[§]Department of Computer Science, Dartmouth College, NH. Email: rangak@cs.dartmouth.edu. Work supported by NSF Grant IIS-0916565 and by Dartmouth College startup funds.

1. INTRODUCTION

We prove matching asymptotic lower and upper bounds on the time-space tradeoff for *mutual exclusion* [7], a fundamental problem in distributed computing. The problem models the situation when $n \geq 2$ asynchronous processes share a resource that may only be accessed by one process at a time. Specifically, each process p_i repeatedly cycles through four sections of code—*Remainder section*, where p_i stays as long as it is not interested in the resource; *Try section*, where p_i competes with other processes for the resource; *Critical section (CS)*, where p_i has exclusive access to the resource; and *Exit section*, where p_i gives up its access so some other process in the Try section may enter the CS. Initially, all processes are in the Remainder section. The *mutual exclusion problem* is to design the code for the Try and Exit sections so that the following two properties hold in all runs of the algorithm: (i) Mutual Exclusion: at most one process is in the CS at any time; and (ii) Starvation Freedom: every process in the Try section eventually enters the CS and every process in the Exit section eventually enters the Remainder section, under the assumption that no process stops taking steps in the Try and Exit sections and no process stays in the CS forever.

For time complexity, the standard practice is to count "remote memory references" (RMRs), which we now explain. In a Distributed Shared Memory (DSM) multiprocessor, shared memory is partitioned among processes. An operation on a shared variable X by a process p_i is considered an RMR if X resides in the partition of some other process $p_j \neq p_i$. In a Cache-Coherent (CC) multiprocessor, each process has a cache, and an operation is considered an RMR if and only if the operation is an update operation or it is a read operation by p_i and X is not in p_i's cache at the time of the operation. The *RMR complexity* of a mutual exclusion algorithm is the worst case number of RMRs that a process makes when executing the Try and Exit sections once. The *space complexity* of an algorithm is the number of shared variables that the algorithm employs.

1.1 Space-RMR tradeoff

For DSM machines, the space complexity of mutual exclusion is $\Omega(n)$[1] and there exist algorithms that achieve $O(n)$ space complexity and $O(1)$ RMR complexity [5, 15]. Thus,

[1]This is because if one process p_i is in the CS while the remaining $n-1$ processes are busywaiting in their Try section, each waiting process p_j must necessarily busywait on a shared variable in its own partition; otherwise, the RMR complexity of the algorithm will be unbounded.

optimal space and optimal RMR are simultaneously achievable for DSM. For CC machines, on the other hand, the landscape is very different: at one end of the spectrum is Anderson's algorithm with $O(1)$ RMR complexity and $O(n)$ space complexity [1], and at the other end exists a simple algorithm with $O(n)$ RMR complexity and $O(1)$ space complexity,[2] giving rise to many interesting questions. For instance, how much space is necessary to achieve constant RMR complexity? With constant space, what is the best RMR complexity that can be achieved? To the best of our knowledge, such interrelationship between space and RMR complexity has never been explored before. In this paper we prove asymptotically matching lower and upper bounds relating RMR and space complexity of mutual exclusion to answer the above questions.

1.2 The bin-pebble game

Our lower and upper bounds for mutual exclusion are derived using a novel bin-pebble game: we show that mutual exclusion has an efficient solution if and only if the game has an efficient solution, and prove tight lower and upper bounds for the game.

The *bin-pebble game* is a single player game, where there are m bins, numbered $1, 2, \ldots, m$. Initially, the first bin has n pebbles and all other bins are empty. The player plays the game in *steps*. In each step, three things happen: (i) the player chooses any one non-empty bin b and shakes it; (ii) the shake causes exactly one pebble in b to "evaporate"; and (iii) for every other pebble in b, the player decides whether the pebble remains in b or moves to a different bin; in the latter case, the player decides which bin the pebble moves to. The cost of the step is the number of pebbles in b just before the shake. The game ends once all n pebbles have evaporated, i.e., after exactly n steps when all bins are empty. The cost incurred by the player is the sum of the costs of the n steps. The goal for the player is to minimize this cost.

Formally, the *n-pebble, m-bin game* is defined as follows. A *configuration* is an m-tuple $(n_1, \ldots, n_m)$ such that each n_i is a non-negative integer (denoting the number of pebbles in bin i) and $0 \leq \sum n_i \leq n$. A *step* is a pair (D, D') of configurations such that if $D = (n_1, \ldots, n_m)$ and $D' = (n'_1, \ldots, n'_m)$, then $\sum n'_i = \sum n_i - 1$ (because exactly one pebble evaporates) and $|\{i \mid n'_i \geq n_i\}| = m - 1$ (because bins that are not shaken don't lose any pebbles and might only gain some). Let i be the unique bin such that $n'_i < n_i$. We say each pebble in bin i *experiences a hit* in the step and define the *cost of the step* as n_i, the number of pebbles experiencing a hit in the step. A *strategy* $\mathcal{S}$ is a sequence of steps $(D_0, D_1), (D_1, D_2), \ldots, (D_{n-1}, D_n)$ such that $D_0 = (n, 0, 0, \ldots, 0)$ is the initial configuration and $D_n = (0, 0, \ldots, 0)$ is the end configuration. Let $hits_{\mathcal{S}}(i, n, m)$ be the number of hits experienced by pebble i in $\mathcal{S}$. The *cost of the strategy* $\mathcal{S}$, denoted $\textsc{total-hits}_{\mathcal{S}}(n, m)$, is the sum of the costs of the steps of $\mathcal{S}$, which is the same as $\sum_i hits_{\mathcal{S}}(i, n, m)$. We use two other cost measures for a strategy $\mathcal{S}$: (i) $\textsc{wc-hits}_{\mathcal{S}}(n, m)$, which is the maximum number of hits that an individual pebble experiences, i.e., $\max_i\{hits_{\mathcal{S}}(i, n, m)\}$, and (ii) $\textsc{amortized-hits}_{\mathcal{S}}(n, m) = \textsc{total-hits}_{\mathcal{S}}(n, m)/n$, which is the average number of hits per pebble. Finally, $\textsc{wc-hits}(n, m)$ and $\textsc{amortized-hits}(n, m)$ are the minimum, over all strategies $\mathcal{S}$, of $\textsc{wc-hits}_{\mathcal{S}}(n, m)$ and $\textsc{amortized-hits}_{\mathcal{S}}(n, m)$, respectively, and are called the *worst case hit complexity* and the *amortized hit complexity* of the n-pebble, m-bin game.

The game is trivial to analyze when there is only one bin: there is only one strategy—repeatedly shake the bin until all pebbles evaporate. Since the last pebble to evaporate experiences n hits, $\textsc{wc-hits}(n, 1) = n$. By how much does the complexity go down if we have more bins? To gain some intuition, suppose we have two bins. We can establish an upper bound of $O(n^{1/2})$ for $\textsc{wc-hits}(n, 2)$ by performing the following two actions repeatedly in each round until all pebbles evaporate: (i) shake the first bin and move $\sqrt{n}$ pebbles to the second bin, and (ii) shake the second bin repeatedly until it is empty. A natural generalization of this strategy gives, for any constant $m \geq 1$, $\textsc{wc-hits}(n, m) = O(n^{1/m})$. Interestingly, we show that this strategy is essentially optimum. In particular, we prove a lower bound that $\textsc{wc-hits}(n, m) = \Theta(n^{1/m})$, for any constant m.

For further intuition, let us ask when we can achieve $O(1)$ for $\textsc{wc-hits}(n, m)$. It is trivially achieved with $m = n - 1$ bins: shake the first bin and move the $n - 1$ remaining pebbles so there is exactly one pebble per bin; then, shake the bins one after the other. Do fewer bins suffice? Perhaps surprising at first, it turns out that the answer is yes. For instance, if we have $2\sqrt{n} - 1$ bins, we can use the following strategy. Divide the n pebbles into $\sqrt{n}$ groups, each group containing $\sqrt{n}$ pebbles. After the initial shake of the first bin and one pebble evaporates, move each of $\sqrt{n} - 1$ groups to different bins and distribute the remaining $n - 1 - (\sqrt{n} - 1)\sqrt{n} = \sqrt{n} - 1$ pebbles to the remaining $\sqrt{n} - 1$ bins, one per bin. In subsequent steps, empty these single pebble bins. Then, shake any bin that has $\sqrt{n}$ pebbles, once more distributing its remaining $\sqrt{n} - 1$ pebbles into the empty bins, ensuring each empty bin gets at most one pebble. Repeat the above actions until all pebbles evaporate. With this strategy, each pebble experiences at most three hits (once in bin 1, once in bin i for $i \in \{2, \ldots, \sqrt{n}\}$, and once in bin i for $i \in \{\sqrt{n}+1, 2\sqrt{n}-1\}$); hence, $\textsc{wc-hits}(n, 2\sqrt{n}) = O(1)$. Again, a natural generalization of this strategy gives $\textsc{wc-hits}(n, n^{\epsilon}) = O(1)$ for any real constant $\epsilon > 0$. The difficult question is whether we can do even better. The answer is no: we prove a lower bound that $\textsc{wc-hits}(n, m) = \Omega(\log n / \log m)$.

1.3 Cost measures for mutual exclusion

Let $\mathcal{A}$ be a mutual exclusion algorithm for n processes using m shared variables. Let R be a finite run of $\mathcal{A}$, r_i be the number of RMRs process i incurs in R, and t_i be the number of times process i executes the Try section in R. Define $\textsc{amortized-rmr}_{R,\mathcal{A}}(n, m)$ as r/t, where $r = \sum_i r_i$ and $t = \sum_i t_i$; and define $\textsc{wc-rmr}_{R,\mathcal{A}}(n, m)$ as the maximum number of RMRs that a process incurs in R in order to execute the Try and the Exit sections once. The *amortized RMR complexity of* $\mathcal{A}$, denoted $\textsc{amortized-rmr}_{\mathcal{A}}(n, m)$, is the maximum of $\textsc{amortized-rmr}_{R,\mathcal{A}}(n, m)$, over all runs R of $\mathcal{A}$. The *amortized RMR complexity of mutual exclusion*, denoted $\textsc{amortized-rmr}(n, m)$, is the minimum, over all algorithms $\mathcal{A}$, of $\textsc{amortized-rmr}_{\mathcal{A}}(n, m)$. The *worst-*

[2]The algorithm keeps two counters *tryCnt* and *exitCnt*, both initialized to 0. In the Try section, a process performs v = fetch&add(*tryCnt*, 1) and then busywaits until *exitCnt*'s value becomes v. In the Exit section, the process increments *exitCnt* by 1.

case RMR complexity of mutual exclusion ($\text{WC-RMR}(n, m)$) is similarly defined from $\text{WC-RMR}_{R,\mathcal{A}}(n, m)$.

1.4 Summary of results

Mutual exclusion is a fundamental problem in concurrent computing, and the design of small RMR complexity algorithms has received a lot of attention in the last two decades. In this paper, we establish for the first time tight bounds relating space and RMR complexity for this problem, for the Cache-Coherent (CC) model. The results are summarized as follows.

- Lower Bounds for Mutual Exclusion
  - $\text{AMORTIZED-RMR}(n, m) \geq \text{AMORTIZED-HITS}(n, m)$ (Theorem 2.1)
  - $\text{AMORTIZED-HITS}(n, m) = \Omega(\max\{n^{1/m}, \frac{\log n}{\log m}\})$ (Theorem 3.1)

  Hence, the lower bound for mutual exclusion is:
  $\text{AMORTIZED-RMR}(n, m) = \Omega(\max\{n^{1/m}, \frac{\log n}{\log m}\})$
- Upper Bounds for Mutual Exclusion
  - $\text{WC-RMR}(n, m) \leq 3\text{WC-HITS}(n, m-2)$ (Theorem 4.2)
  - $\text{WC-HITS}(n, m) = O(mn^{1/m})$ when $m = O(\log n)$ (Theorem 5.1)
  - $\text{WC-HITS}(n, m) = O(\frac{\log n}{\log m})$ when $m = \Omega(\log^{1+\epsilon} n)$ for any real constant $\epsilon > 0$ (Theorem 5.2)

  Hence, upper bounds for mutual exclusion are:
  - $\text{WC-RMR}(n, m) = O(mn^{1/(m-2)})$ when $m = O(\log n)$
  - $\text{WC-RMR}(n, m) = O(\frac{\log n}{\log m})$ when $m = \Omega(\log^{1+\epsilon} n)$ for any real constant $\epsilon > 0$

Our lower bounds are strong in the sense that they hold regardless of how strong and large the hardware objects are. The upper bounds would be similarly strong if our algorithms used only read and write operations. However, because of Attiya et al's result that $o(\log n)$ RMR complexity is impossible using only read and write operations [2], any algorithm must necessarily employ stronger operations in order to realize upper bounds that match or nearly match our lower bounds, which are sub-logarithmic whenever algorithms are allowed at least $\log n$ space. Our algorithms are designed to use the fetch&increment operation.

1.5 Highlights of our contributions

- Our bounds imply that subpolynomial RMR complexity is impossible using constant space and constant RMR complexity is impossible using subpolynomial space, no matter how strong the hardware operations are.
- Our lower and upper bounds for mutual exclusion are matching for all but small values of m: $\text{AMORTIZED-RMR}(n, m)$ and $\text{WC-RMR}(n, m)$ are both $\Theta(\frac{\log n}{\log m})$ when $m = \Omega(\log^{1+\epsilon} n)$, for any real constant $\epsilon > 0$.
- Mutual exclusion can be a "messy" problem to analyze because of system details such as asynchrony and cache coherence. But we are able to design a purely combinatorial game that abstracts out all system details and yet exactly captures the complexity of mutual exclusion.

1.6 Related work

Dijkstra was the first to formulate and solve the n-process mutual exclusion problem [7]. Stronger properties (e.g., starvation-freedom [12], first-come-first-served [13], self-stabilization [14], etc.) were subsequently defined and realized (see the survey [16]). These early algorithms were unsuitable for multiprocessors because of their high RMR complexity. Subsequent research led to the design of "local spinning" algorithms that achieved constant RMR complexity using strong hardware operations, such as fetch&add or swap [1, 5, 10, 15]. Cypher [6] proved the necessity of such operations: he showed that constant RMR complexity is unattainable using only read/write and conditional operations, such as compare&swap. Yang and Anderson [17] showed that $O(\log n)$ RMR complexity is achievable for n-process mutual exclusion using read/write operations, while Fan and Lynch [8] and Attiya, Hendler, and Woelfel [2] proved a matching lower bound of $\Omega(\log n)$. Hendler and Woelfel [11] and Bender and Gilbert [3] have recently beaten this bound with the help of randomization. The space complexity of mutual exclusion, when using only read/write shared variables, was resolved by Burns and Lynch [4] who showed that $\Theta(n)$ shared variables are necessary and sufficient to (deterministically) solve mutual exclusion.

Unlike the above works where time or space complexity was studied in isolation, this paper investigates how time trades off with space when solving mutual exclusion deterministically.

We prove the lower bounds using the potential method. While potential functions are commonly used for establishing upper bounds (e.g., on running time of algorithms and data structures or on competitive ratio of online algorithms), their use for proving lower bounds is more subtle and rare. We know of only one other lower bound in distributed computing that is proved using the potential method [9].

2. REDUCING THE BIN-PEBBLE GAME TO ONE-SHOT MUTUAL EXCLUSION

One-shot mutual exclusion is a restricted version of the mutual exclusion problem where each process executes the Try, Critical, and Exit sections at most once and terminates upon completing the Exit section. If there is a low cost algorithm that uses m shared variables to solve n-process one-shot mutual exclusion, we show that there is a low cost bin-pebble strategy for n pebbles using m bins.

Let $\mathcal{A}$ be a one-shot mutual exclusion algorithm for a set Π of n processes and V be the set of shared variables used in $\mathcal{A}$. A *configuration* (of the algorithm $\mathcal{A}$) represents the state of $\mathcal{A}$ at a point in time, and is specified by the states of the n processes and the values of the shared variables in V at that time. In the *initial configuration*, each process is in the remainder section and each variable holds the initial value prescribed by $\mathcal{A}$. The configuration changes when a process takes a *step*: in a step, a process p performs an operation on a shared variable $x \in V$, thereby changing p's own state and possibly x's state. A *run from a configuration C* is a sequence of configurations $C_0, C_1, \ldots$ such that $C_0 = C$ and, for all $i \geq 1$, C_i results when some process executes a step from C_{i-1}.

Let σ denote a step by a process p in a run R from the initial configuration. We say *p incurs an RMR on x in σ*, or *σ incurs an RMR*, if one of the following two conditions holds: (i) p performs an update operation (i.e., a non-read operation) on x in σ, or (ii) p performs a read on x in σ and either σ is the first step in R where p reads x or some other process updates x after p's latest read of x before σ. The *total RMR cost of R* is the number of steps in R that incur an RMR. The *worst-case total RMR cost of algorithm $\mathcal{A}$* is the maximum, over all runs R of $\mathcal{A}$, of the total RMR cost of R.

A run is a *solo-run of process p* if all steps in that run are by p. We say a *process p spins on a set $V' \subseteq V$ of variables in a configuration C* if the following conditions hold in the infinite solo-run R of p from C: (i) in every step of R, the operation performed by p is a read on a variable in V' and does not incur an RMR, and (ii) for all $x \in V'$, there are infinitely many steps of R in which p reads x. We call a variable $x \in V$ a *spin variable* if there are $p \in \Pi$, $V' \subseteq V$, and reachable configuration C such that $x \in V'$ and p spins on V' in C. We say *p spins in C* if there exists $V' \subseteq V$ such that p spins on V' in C.

THEOREM 2.1. *Suppose there is a one-shot starvation-free mutual exclusion algorithm $\mathcal{A}$ for n processes that uses m spin variables and has a worst-case total RMR cost of t. Then, there is a bin-pebble strategy of cost at most t for n pebbles using m bins.*

Proof: The high level ideas in the design of the strategy are as follows. Each bin corresponds to a spin variable and, roughly speaking, each pebble corresponds to a process. We consider a special, high cost run R of the algorithm $\mathcal{A}$ and focus on a certain subsequence $C_0, C_1, \ldots, C_n$ of configurations in this run. We then define a corresponding sequence $D_0, D_1, \ldots, D_n$ of bin-pebble configurations such that there are at most $n - i$ pebbles left in D_i and it is possible to go from D_i to D_{i+1} with a cost that is at most the number of RMRs incurred in R when going from C_i to C_{i+1}. Establishing these correspondences is the crux of the proof, which we now present.

The run R, the configurations $C_0, C_1, \ldots, C_n$, and process names $p_1, p_2, \ldots, p_n$ are defined in Figure 1 and their properties are stated in the following lemma.

LEMMA 2.2.

1. *R is a run of algorithm $\mathcal{A}$ from the initial configuration C_0; $C_0, C_1, \ldots, C_n$ is a subsequence of R; for all $1 \le i \le n$, in configuration C_i, p_i is in the CS, processes $p_{i+1}, \ldots, p_n$ are spinning in the Try section, and processes $p_1, \ldots, p_{i-1}$ are in the Exit section or have terminated the protocol.*

2. *Each of the n processes incurs an RMR in the fragment of R from C_0 to C_1.*

3. *For all $2 \le i \le n$, process p_i incurs an RMR in the fragment of R from C_{i-1} to C_i.*

Proof Sketch: Parts (1) and (2) are immediate from the construction in Figure 1. Part (3) follows from the observation that, since p_i spins in C_{i-1} and is in the CS in C_i, p_i must incur an RMR (by reading an updated spin variable) in the fragment of the run from C_{i-1} to C_i. ■

The definitions below and the lemma that follows will help extract a bin-pebble strategy.

- Let $x_1, x_2, \ldots, x_m$ denote the set of spin variables used in the algorithm $\mathcal{A}$, and, for $1 \le i \le n-1$, let V_i denote the set of spin variables that are updated in the fragment of the run R from C_i to C_{i+1}.

- For $1 \le i < j \le n$, define $\textsc{var}(i,j) = k$ if, in the fragment of R starting from C_i, x_k is the variable on which p_j first incurs an RMR. (Part 3 of Lemma 2.2 implies that $\textsc{var}(i,j) = k$ is well defined.)

- For $1 \le i \le n$ and $1 \le k \le m$, define $\textsc{procs}(i,k) = \{j \mid j > i \text{ and } \textsc{var}(i,j) = k\}$, which is the set of all processes whose first RMR, in the fragment of R starting from C_i, is on the variable x_k.

LEMMA 2.3.

1. *For all $1 \le i \le n-1$, the total number of RMRs incurred in the fragment of the run R from C_i to C_{i+1} is at least $\sum_{x_k \in V_i} |\textsc{procs}(i,k)|$.*

2. *For all $1 \le i \le n-1$ and $x_k \notin V_i$, $\textsc{procs}(i,k) \subseteq \textsc{procs}(i+1,k)$.*

3. *For all $1 \le i \le n$, $\sum_{k \in V} |\textsc{procs}(i,k)| = n - i$.*

Proof Sketch: In the fragment of the run R from C_i to C_{i+1}, a process p_j, $j > i$, incurs an RMR if and only if $\textsc{var}(i,j) = k$ for some $x_k \in V_i$. Parts (1) and (2) of the lemma follow from this observation and definitions. Part (3) follows from the observation that $\{p_{i+1}, p_{i+2}, \ldots, p_n\}$ is exactly the set of processes in the Try section in C_i and, for each p_j in this set, there is a unique k, $1 \le k \le m$, such that p_j appears in $\textsc{procs}(i,k)$. ■

In Figure 2 , we exploit the above lemma to construct from R a bin-pebble strategy for n pebbles using m bins, and argue its correctness. The next lemma uses the invariant stated there to bound the cost of the strategy.

LEMMA 2.4. *For all $i \ge 0$, the cost of the steps that take the bin-pebble configuration from D_i to D_{i+1} is at most the number of RMRs in the fragment of run R from C_i to C_{i+1}.*

Proof Sketch: The cost of the step that takes the bin-pebble configuration from D_0 to D_1 is exactly n, which is at most the number of RMRs in the fragment of run R from C_0 to C_1 (by Part (2) of Lemma 2.2). For $i > 0$, the total cost of the steps that take the configuration from D_i to D_{i+1} is

$$\begin{aligned}
\sum_{k \in V_i'} np(i,k) &\le \sum_{k \in V_i} np(i,k) \\
&\qquad \text{(since } V_i' \subseteq V_i\text{, see Line 4 of Figure 2)} \\
&\le \sum_{k \in V_i} |\textsc{procs}(i,k)| \\
&\qquad \text{(by the invariant } \textsc{inv}(i) \text{ stated in Figure 2)} \\
&\le \text{ RMRs in the fragment of } R \text{ from } C_i \text{ to } C_{i+1} \\
&\qquad \text{(by Part (1) of Lemma 2.3)} \ ■
\end{aligned}$$

It is immediate from the above lemma that the cost of the bin-pebble strategy in Figure 2 is at most the total number of RMRs in the run R. This concludes the proof of Theorem 2.1. ■

```
1.    $P \leftarrow \Pi$, where $\Pi = \{1, 2, \ldots, n\}$ is the set of processes.
2.    From the initial configuration $C_0$, processes in $P$ take one step each (in any order),
      thereby ensuring that each process leaves the remainder section and incurs an RMR.
3.    for $i = 1$ to $n$
          INVARIANT: Each process in $P$ is in the Try section and each process in $\Pi - P$ is
                     in the Exit section or has terminated.
4.        Processes in $\Pi$ that have not terminated take steps in a round-robin order until
          some process in $P$ enters the CS, which is guaranteed by the Starvation Freedom property.
          Let $p_i$ denote the process that enters the CS.
5.        With $p_i$ in the CS, processes $P - \{p_i\}$ in the Try section take steps in a round-robin order
          until a configuration is reached where every process in $P - \{p_i\}$ spins. (Since the RMR cost
          is bounded, such a configuration is necessarily reached.)
          Let $C_i$ denote this configuration.
6.        Process $p_i$ leaves the CS.
7.        $P \leftarrow P - \{p_i\}$
```

Figure 1: The definition of run R

3. LOWER BOUNDS FOR THE BIN-PEBBLE GAME

In this section, we present our lower bounds on the amortized hit complexity of bin-pebble game. Our main result is the following:

THEOREM 3.1. *For any $n > 0, m \geq 1$,*

$$\text{AMORTIZED-HITS}(n, m) = \Omega\left(\max\left(\frac{\log n}{\log(2m)}, n^{1/m}\right)\right)$$

This theorem, together with Theorem 2.1, implies the following lower bound for amortized RMR complexity of mutual exclusion:

THEOREM 3.2. *For any mutual exclusion algorithm $\mathcal{A}$ for $n \geq 2$ processes using $m \geq 1$ spin variables,*

$$\textit{Amortized RMR complexity of } \mathcal{A} = \Omega\left(\max\left(n^{1/m}, \frac{\log n}{\log(2m)}\right)\right) \quad (1)$$

To prove Theorem 3.1, we will show two bounds:

$$\text{TOTAL-HITS}(n, m) = \Omega\left(\frac{n \log n}{\log(2m)}\right) \quad \text{For any } n > 0, m \geq 1. \quad (2)$$

$$= \Omega\left(n^{\frac{m+1}{m}}\right) \quad \text{For } m = O(\log n). \quad (3)$$

We note that the first lower bound (2) holds for any m. The second bound (3) holds for only for $m = O(\log n)$ and is stronger than (2) when $m = O(\log n/\log\log n)$. Both these bounds are proved using potential functions. Roughly speaking, we show that the amortized cost per shake of a bin is $\Omega(\log n/\log m)$ for (2) and $\Omega(n^{1/m})$ for (3). The potential functions we design for these two bounds are qualitatively very different, and are also used in very different ways. Before we describe the formal proof, we give some overview and intuition for these potential functions.

Intuition for the Potential Functions:.

The main idea for proving (2) is the following: We define a potential function Φ based on the state (how pebbles are distributed in bins). Initially, for state $(n, 0, \ldots, 0)$, Φ has value $\Omega(n \log n/\log m)$, and it is 0 when all pebbles have evaporated eventually. The function Φ is constructed so that no matter how the algorithm redistributes its pebbles upon shaking a bin at time t, and no matter what the current state is, the decrease in Φ is no more than the cost incurred by the algorithm. That is, for any $0 \leq t \leq n-1$:

$$w(t) \geq \Phi(t) - \Phi(t+1),$$

where $\Phi(t)$ is the potential at t, and $w(t)$ is the cost of the algorithm at t. Summing up over t, we get that the total cost is $\sum_t w(t) \geq \Phi(0) - \Phi(n) = \Omega(n \log n/\log m)$, implying (2).

The choice of Φ is guided by the best algorithmic strategy for the bin-pebble game. Recall that we need Φ to not decrease by more than the cost $w(t)$, no matter what the algorithm does. It turns out that the most powerful step for the algorithm is to redistribute the pebbles from the bin it shakes. That is, upon shaking a bin with x pebbles it redistributes x/m pebbles in the m bins (not surprisingly, our close-to-optimum algorithm in section 5 also roughly does the same). So, to define Φ we need a function f such that for the above step, the change in the potential, $f(x) - mf(x/m)$, should be about x, the cost incurred by the algorithm. This holds for the entropy-like function $f(x) = x \log x/\log m$, and indeed, this is how we define Φ (in (5)).

To prove (3), we adopt a very different and a much more subtle approach. We only consider the first $n/2$ time steps, and show that any algorithm incurs a cost of $\Omega(n^{1/m})$ per time step in an amortized sense. More precisely, we design a Φ such that for each time step $t \in [0, n/2 - 1]$, no matter which bin the algorithm shakes, and how it redistributes the pebbles, and what the current state is at time t, it holds that

$$w(t) + \Phi(t+1) - \Phi(t) \geq (1/8)n^{1/m}. \quad (4)$$

Moreover $\Phi(t)$ always stays in the range $[0, n/2]$. Summing inequality above from $t = 0$ to $n/2 - 1$ implies that total cost is $\sum_t w(t) \geq (1/16)n^{1+1/m} + \Phi(0) - \Phi(n/2) = \Omega(n^{1+1/m})$, giving the claimed lower bound.

Intuitively (4) says that if an algorithm tries to get away with low cost by shaking a bin with $\ll n^{1/m}$ pebbles at some step, then the "state" of the algorithm gradually worsens and eventually it will have to shake a bin with $\gg n^{1/m}$

The strategy is specified via a sequence $D_0, D_1, \ldots, D_n$ of bin-pebble configurations. Let $np(i,k)$ denote the number of pebbles in bin k in configuration D_i. We ensure the following key invariant, for all $1 \leq i \leq n$:
Invariant $\textsc{inv}(i)$: For all $k \in V$, $np(i,k) \leq |\textsc{procs}(i,k)|$

1. D_0 is the configuration where all n pebbles are in bin 1.
2. D_1 is the configuration obtained by shaking bin 1 and distributing the $n-1$ pebbles (that are left after the shake) among the k bins so as to ensure $\textsc{inv}(1)$. Such a distribution is possible since $\sum_{k=1}^{m} |\textsc{procs}(1,k)| = n-1$.
3. **for** $i = 1$ **to** $n-1$
4. $V_i' = \{k \,:\, |\textsc{procs}(i+1,k)| < np(i,k)\}$

       We observe that $V_i' \subseteq V_i$ since

$$\begin{aligned}
k \in V_i' &\Rightarrow |\textsc{procs}(i+1,k)| < np(i,k) && \\
&\Rightarrow |\textsc{procs}(i+1,k)| < |\textsc{procs}(i,k)| && \text{(by } \textsc{inv}(i)\text{)} \\
&\Rightarrow k \in V_i && \text{(by Part (2) of Lemma 2.3)}
\end{aligned}$$

5.   **if** $V_i' = \emptyset$
6. $D_{i+1} = D_i$

           (Since $np(i+1,k) = np(i,k) \leq |\textsc{procs}(i+1,k)|$, we have $\textsc{inv}(i+1)$.)
7.   **else**
8.   Let $V_i' = \{k_1, k_2, \ldots, k_{|V_i'|}\}$
9.   Obtain D_{i+1} from D_i by performing a sequence of $|V_i'|$ steps where, in the jth step, $1 \leq j \leq |V_i'|$, bin k_j is shaken. Of the $np(i,k_j)$ pebbles that are in this bin before the shake, one pebble evaporates; $|\textsc{procs}(i+1,k_j)|$ are retained in the bin; and the remaining $np(i,k_j) - 1 - |\textsc{procs}(i+1,k_j)|$ pebbles are transferred to the bins $V - V_i'$, where V denotes $\{1, 2, \ldots, k\}$.

           We claim that this transfer from the bins in V_i' to the bins in $V - V_i'$ can be done in a way that $\textsc{inv}(i+1)$ holds. Below we justify this claim by showing that the total number $\textsc{ntransfer}$ of pebbles to transfer from bins in V_i' is at most $\textsc{rleft}$, the room left in the bins in $V - V_i'$ to absorb more pebbles:

$$\begin{aligned}
\textsc{ntransfer} &= \sum_{k \in V_i'} (np(i,k) - 1 - |\textsc{procs}(i+1,k)|) && \\
&\leq -1 + \sum_{k \in V_i'} np(i,k) - \sum_{k \in V_i'} |\textsc{procs}(i+1,k)| && \text{(since } V_i' \neq \emptyset\text{)} \\
&= -1 + \sum_{k \in V} np(i,k) - \sum_{k \in V - V_i'} np(i,k) - \sum_{k \in V_i'} |\textsc{procs}(i+1,k)| && \text{(rearranging terms)} \\
&\leq -1 + \sum_{k \in V} |\textsc{procs}(i,k)| - \sum_{k \in V - V_i'} np(i,k) - \sum_{k \in V_i'} |\textsc{procs}(i+1,k)| && \text{(using } \textsc{inv}(i)\text{)} \\
&= \sum_{k \in V} |\textsc{procs}(i+1,k)| - \sum_{k \in V - V_i'} np(i,k) - \sum_{k \in V_i'} |\textsc{procs}(i+1,k)| && \text{(by Part (3) of Lemma 2.3)} \\
&\leq \sum_{k \in V - V_i'} (|\textsc{procs}(i+1,k)| - np(i,k)) \;=\; \textsc{rleft} && \text{(rearranging terms)}
\end{aligned}$$

    end-for

It follows from $\textsc{inv}(n)$ and Part (3) of Lemma 2.3 that, for all k, $np(n,k) \leq |\textsc{procs}(n,k)| = 0$.
Hence, all bins are empty in D_n, and the sequence $D_0, D_1, \ldots, D_n$ defined above is indeed a valid strategy.

Figure 2: Using R to construct a bin-pebble strategy for n pebbles using m bins

pebbles and incur a lot of cost. The function Φ captures this worsening of the algorithm's state, and whenever the algorithm incurs low cost, Φ contributes to make the overall cost $\Omega(n^{1/m})$, but then recharges itself appropriately if the algorithm incurs high cost. The intuition for this Φ is harder to describe (and it is perhaps easier to just read the analysis), but roughly, it measures how far the state of algorithm is from the state of the close-to-optimum strategy for the case of $O(1)$ bins that we outlined earlier.

Proof of theorem 3.1. If $m = 1$, then the theorem clearly holds as $\text{TOTAL-HITS}(n, m) = n(n + 1)/2$. Henceforth, we will assume that $m \geq 2$, and hence in particular we have $\log m > 0$. We first prove equation 2. Let $\mathcal{S}$ be a strategy for the bin-pebble game with n pebbles and m bins. Let $n_i(t)$ denote the size of bin i at tth step of $\mathcal{S}$. We define the potential

$$\Phi(t) = \frac{1}{4\log m}\sum_{i=1}^{m} n_i(t)\log(n_i(t)+1). \tag{5}$$

Here log the logarithm is with respect base e. By the rules of game we have, in a step,

1. One pebble evaporates, i.e. $\sum_i n_i(t+1) = (\sum_i n_i(t)) - 1$.

2. A cost of $n_i(t)$ is incurred where i is the index of the shaken bin.

Let $w(t)$ denote the cost incurred at time t. As stated previously, it suffices to show that for each step, the decrease in potential is at most the actual cost of the step.

Claim 3.3. *For all $0 \leq t \leq n-1$, it holds that $w(t) + \Phi(t+1) - \Phi(t) \geq 0$.*

We first observe a few easy properties of the function $f(x) = x\log(x + 1)$.

Observation 3.4. *The function $f(x) = x\log(x + 1)$ satisfies the following properties.*

1. Clearly, $f(x) \geq 0$ for $x \geq 0$. It is increasing for $x > 0$ as $f'(x) = \log(x + 1) + x/(x + 1) > 0$. Moreover, f is convex as $f''(x) = d/dx(f'(x)) = 1/(x + 1) + 1/(x + 1)^2 \geq 0$ for $x \geq 0$.

2. Since f is convex, for any $x_1, \ldots, x_k \geq 0$, we have $\sum_i f(x_i) \geq kf(\sum_i x_i/k)$.

3. Since f is convex, for any $x > y$, $f(x) - f(y) \leq (x - y)f'(x)$. Moreover, for any $\delta > 0$, we have $f(x + \delta) - f(x) \geq f(y + \delta) - f(y)$.

4. Let $x \geq y > 0$. As $\log a \leq a - 1$ for any $a \geq 1$, and setting $a = x/y$ we obtain $\log(x/y) \leq (x/y) - 1$, which can be written as $y(\log x) + y \leq x + y\log y$.

We are now ready to prove the claim.

Proof of Claim 3.3. Let i be the bin that is shaken at time t i.e., $n_i(t + 1) < n_i(t)$. So, $w(t) = n_i(t)$. Let $\beta = n_i(t) - n_i(t+1)$ denote the decrease in the pebble count for bin i. For $j \neq i$, let $\gamma_j = n_j(t + 1) - n_j(t)$ denote the increase in pebble count for bin j. Since exactly one pebble evaporates $\sum_{j\neq i}\gamma_j = \beta - 1$.

When the configuration changes from t to $t+1$, contribution to Φ of bins $j \neq i$ increases and the contribution to Φ of the bin i decreases. We will bound these changes suitably. First, consider the increase in contribution to Φ of bins $j \neq i$. This is equal to $\sum_{j\neq i} f(n_j(t + 1)) - f(n_j(t))/(4\log m)$. We have

$$\begin{aligned}\sum_{j\neq i} f(n_j(t+1)) - f(n_j(t)) &= \sum_{j\neq i} f(n_j(t)+\gamma_j) - f(n_j(t))\\ &\geq \sum_{j\neq i}(f(\gamma_j) - f(0)) = \sum_{j\neq i} f(\gamma_j),\end{aligned}$$

where the inequality follows from Observation 3.4(part 3). By convexity of f, i.e., Observation 3.4(part 2) and setting $\gamma_i = 0$, we get

$$\begin{aligned}\sum_{j\neq i} f(\gamma_j) &= \sum_{j=1}^{m} f(\gamma_j) \geq mf\left(\sum_{j=1}^{m}\gamma_j/m\right) = mf((\beta-1)/m)\\ &= (\beta-1)\log(((\beta-1)/m)+1) \geq (\beta-1)\log(\beta/m).\end{aligned}$$

Therefore, the increase in potential due to bins $j \neq i$ is

$$\frac{1}{4\log m}\sum_{j\neq i} f(n_j(t{+}1)) - f(n_j(t)) \geq \frac{1}{4\log m}\left((\beta-1)\log(\beta/m)\right). \tag{6}$$

Now consider the ith bin. We will show that the decrease in potential due to it satisfies

$$\frac{1}{4\log m}\left(f(n_i(t)) - f(n_i(t+1))\right) \leq \frac{1}{4\log m}(2w(t){+}\beta\log\beta). \tag{7}$$

This follows as, by Observation 3.4 (part 3),

$$\begin{aligned}f(n_i(t)) - f(n_i(t+1)) &\leq (\beta)f'(n_i(t))\\ &= \beta\left(\log(n_i(t)+1) + \frac{n_i(t)}{n_i(t)+1}\right)\\ &\leq \beta(\log(n_i(t)+1)+1)\\ &\leq n_i(t) + 1 + \beta\log\beta \qquad (8)\\ &\leq 2n_i(t) + \beta\log\beta \qquad (9)\\ &= 2w(t) + \beta\log\beta.\end{aligned}$$

Here inequality (8) follows from Observation 3.4 (part 4) with $y = \beta$ and $x = n_i(t) + 1$, and noting that $y \leq x$. Inequality (9) follows because $n_i(t) \geq 1$.

By (6) and (7), we have that the change in potential is

$$\begin{aligned}\Phi(t+1) - \Phi(t) &\geq \frac{1}{4\log m}\left((\beta-1)\log(\beta/m) - 2w(t) - \beta\log\beta\right)\\ &= \frac{1}{4\log m}\left(-(\beta-1)\log m - \log\beta - 2w(t)\right)\\ &\geq \frac{1}{4\log m}\left(-w(t)\log m - w(t) - 2w(t)\right) \qquad (10)\\ &\geq -w(t)\end{aligned}$$

Here (10) follows as $\beta \leq n_i(t) = w(t)$ and $\log\beta \leq \beta \leq w(t)$. Thus, the claim follows. ■

When** $m = O(\log n)$**:

We now show that $\text{TOTAL-HITS}(n, m)$ is at least $\Omega(n^{1+1/m})$. We will assume that $m < (\log n)/3$ (otherwise if $m \geq (\log n)/3$, then $n^{1/m} = 2^{(\log n)/m} \leq 2^3 = 8$, so the bound is trivial).

Let S be a strategy for the bin-pebble game with n pebbles and m bins. We will only consider the first $n/2$ steps and show that the total cost during these steps is $\Omega(n^{1+1/m})$. Let $s_j(t)$ denote the total number of pebbles in the first j lightest bins at time t. We will call the lightest bin 1-

smallest, second lightest bin 2-smallest and so on. Define $h_j = (1/2)n^{j/m}$. Define the potential at time t as

$$\Phi(t) = cn^{1/m} \max_{j=1}^{m}\big(\max(0, h_j - s_j(t))\big), \tag{11}$$

where $c = 1/4$.

Clearly, $\Phi(t) \geq 0$. If $\Phi(t) > 0$, we say that potential is *determined* by j-th smallest bin if j is the smallest index for which the equality holds in (11), i.e. smallest j such that $\Phi(t) = h_j - s_j(t)$. We note a few properties of Φ:

Observation 3.5. *For any t, $\Phi(t) \leq cn/2$.*

Proof. As we only consider first $n/2$ steps, and $s_m(t)$ is simply the total number of pebbles at time t, it holds that $s_m(t) \geq n/2 = h_m$. So, if $\Phi(t) > 0$, it cannot be determined by $j = m$ in (11). As $h_j \leq (1/2)n^{(m-1)/m}$ for $j \leq m-1$, we have $\Phi(t) \leq cn^{1/m}h_{m-1} = cn/2$. ■

Observation 3.6. *If $\Phi(t) = 0$, then each bin has at least $h_1 = (1/2)n^{1/m}$ pebbles.*

Proof. If $\Phi(t) = 0$ then clearly $h_1 - s_1(t) \leq 0$ (in fact $h_j - s_j(t) \leq 0$ for all $1 \leq j \leq m$), and note that $s_1(t)$ is the number of pebbles in the least loaded bin. ■

Observation 3.7. *Suppose $\Phi(t) > 0$ and it is determined by j. Then, $\Phi(t) \leq h_{j+1}/4$ (note that by Observation 3.5, $j \leq m-1$, and hence h_{j+1} is well-defined). Moreover, the number of pebbles in the $(j+1)$-th smallest bin is at least $h_{j+1}/2$.*

Proof. As j determines $\Phi(t)$ we get $\Phi(t) = cn^{1/m}(h_j - s_j(t)) \leq cn^{1/m}h_j = ch_{j+1} = h_{j+1}/4$. For second part, we observe that $h_{j+1} - s_{j+1}(t) \leq h_j - s_j(t)$ (otherwise $j+1$ or a higher index would determine Φ). This implies that $s_{j+1}(t) - s_j(t) \geq h_{j+1} - h_j \geq h_{j+1}/2$ and hence the claim as $s_{j+1}(t) - s_j(t)$ is the number of pebbles in the $(j+1)$-th smallest bin. ■

Let $w(t)$ be the cost incurred in time step t. Let $\Delta\Phi(t) = \Phi(t+1) - \Phi(t)$. Then, following claim holds:

Claim 3.8. *For all $0 \leq t \leq n/2 - 1$, $w(t) + \Delta\Phi(t) \geq (1/8)n^{1/m}$.*

Proof. First, consider the case when $\Phi(t) = 0$. Here Φ cannot decrease further and hence $\Delta\Phi(t) \geq 0$. As $\Phi(t) = 0$, each bin has at least h_1 pebbles (Observation 3.6), so $w(t) \geq h_1$. So, $w(t) + \Delta\Phi(t) \geq h_1 = n^{1/m}/2$.

So, henceforth we assume that $\Phi(t) > 0$, and let us say it is determined by j. We consider two sub-cases depending on the number of pebbles in the bin (call it B) that is shaken. Let i be the rank of bin B when bins are ranked in increasing order of number of pebbles at time t (note that the ranking of bins could be different at time $t+1$), i.e., B is the i-smallest bin at time t.

1. If $i \geq j+1$: Since $\Phi(t)$ is determined by j, we have $\Phi(t) \leq ch_{j+1} = h_{j+1}/4$ (Observation 3.7). Moreover, the $(j+1)$-th smallest bin (and hence the ith smallest bin) has at least $h_{j+1}/2$ pebbles (Observation 3.7). In the worst case, $\Phi(t+1)$ can drop to 0, so $\Delta\Phi \geq -h_{j+1}/4$. Hence, $w(t) + \Delta\Phi(t) \geq h_{j+1}/2 - h_{j+1}/4 = h_{j+1}/4 \geq h_1/4$ and the claim holds.
2. If $i \leq j$: We will show that $\Delta\Phi(t) \geq cn^{1/m}$, which will imply the claim as $w(t) \geq 0$ and $c = 1/4$. By (11), $\Phi(t+1) \geq cn^{1/m}(h_j - s_j(t+1))$, where $s_j(t+1)$ is the total number of pebbles in the j least loaded bins at time $t+1$. Now, let X denote set of the j least loaded bins at time t (these could be different from the j least loaded bins at $t+1$). Let $\tilde{s}_j(t+1)$ denote the total number of pebbles in the bins in X at time $t+1$. Clearly, as $\tilde{s}_j(t+1) \geq s_j(t+1)$,
$$\Phi(t+1) \geq cn^{1/m}(h_j - \tilde{s}_j(t+1)). \tag{12}$$
Since $i \leq j$ by our assumption, we have that the shaken bin B lies in the set X. We claim that
$$\tilde{s}_j(t+1) \leq s_j(t) - 1. \tag{13}$$
This follows because when B is shaken at time t, one pebble evaporates, and none of the bins with indices $j+1$ and higher (i.e. the bins not in X) lose pebbles (they may only gain some extra pebbles from B). Putting these together, we have that
$$\begin{aligned}\Phi(t+1) - \Phi(t) &\geq cn^{1/m}(h_j - \tilde{s}_j(t+1)) - \Phi(t)\\ &= cn^{1/m}(s_j(t) - \tilde{s}_j(t+1)) \geq cn^{1/m},\end{aligned}$$
where the first inequality follows from (12), the second equality follows as $\Phi(t)$ is determined by j, so $\Phi(t) = cn^{1/m}(h_j - s_j(t))$, and the last inequality by (13). So $\Delta\Phi(t) \geq cn^{1/m}$ as claimed.

■

Summing the inequality in Claim 3.8 over $t \in [0, n/2-1]$ and as $\Phi(n/2) - \Phi(0) \in [-cn/2, cn/2]$,

$$\begin{aligned}\text{TOTAL-HITS}_{\mathcal{S}}(n,m) &= \sum_{t=0}^{n-1} w(t)\\ &\geq n^{1+1/m}/16 + \Phi(0) - \Phi(n/2-1)\\ &= \Omega(n^{1+1/m}).\end{aligned}$$

■

4. REDUCING MUTUAL EXCLUSION TO THE BIN-PEBBLE GAME

In this section we show that, given a bin-pebble strategy for n pebbles using m bins, where the maximum number of hits any pebble experiences is t, we can design a mutual exclusion algorithm for n processes that uses $O(m)$ shared variables and has $O(t)$ worst case RMR complexity.

```
TOKEN: a non-spin shared variable, initialized to 0
X[1], X[2], ... X[$m$]: spin variables, initialized to 0

Code for each of the $n$ processes
1. $i$ = fetch&increment(TOKEN, 1)
2. **for** $j = 1$ **to** $s(i)$
3.     **wait till** X[$b(i,j)$] $\geq p(i,j)$
4. CS
5. **if** $i < n-1$ **then** X[$b(i+1, s(i+1))$] $= i+1$
```

Figure 3: One-shot mutual exclusion algorithm, designed using a bin-pebble strategy

```
TOKEN: unbounded non-spin shared variable, initialized to 0
X[1], X[2], ... X[m]: 2n-valued spin variables, initialized to 0
TOGGLE: single-bit spin variable, initialized to 0

Code for each of the n processes
 1.  $t$ = fetch&increment(TOKEN, 1)
 2.  $(parity, i) = t \bmod 2n$
 3.  wait till TOGGLE == $parity$
 4.  for $j = 1$ to $s(i)$
 5.      wait till $(X[b(i,j)] \cdot parity == parity) \wedge (X[b(i,j)] \cdot name \geq p(i,j))$
 6.  CS
 7.  if $(i < n-1)$ then $X[b(i+1, s(i+1))] = (parity, i+1)$
 8.  else
 9.      $X[1] = (\overline{parity}, 0)$
10.      TOGGLE = $\overline{parity}$
```

Figure 4: Long-lived Starvation-free, FCFS mutual exclusion algorithm

First we design a one-shot algorithm and then extend it to be a general algorithm where each process can repeatedly cycle through the Try, Critical, and Exit sections.

Let $\mathcal{B}$ be a bin-pebble strategy for n pebbles using m bins. Assign names from $\Pi = \{0, 1, \ldots, n-1\}$ to the pebbles so that pebbles evaporate in the order of their names. Name the bins $1, 2, \ldots, m$ so that bin 1 contains all n pebbles initially. A *bin-configuration* specifies the set of pebbles that each bin contains. The strategy $\mathcal{B}$ is specified by a sequence $D_0, D_1, \ldots, D_n$, where D_0 denotes the initial bin-configuration and, for $1 \leq i \leq n$, D_i denotes the bin-configuration after i steps. We say *pebble i experiences a hit in step j* if and only if pebble i is present in the bin that is shaken in step j. Let $s(i)$ denote the total number of hits that pebble i experiences. Let $b(i,j)$ denote the bin where pebble i experiences its jth hit, and $p(i,j)$ denote the pebble that evaporates in the step where pebble i experiences its jth hit. Note that $p(i, s(i)) = i$ (since i evaporates as soon as it experiences the $s(i)^{th}$ hit) and $b(i,1) = 1$ (since every pebble is initially in bin 1).

Theorem 4.1. *Suppose there is a bin-pebble strategy $\mathcal{B}$ for a set $\Pi = \{0, 1, \ldots, n-1\}$ of pebbles using m bins, where the maximum number of hits any pebble experiences is t. Then, there is a one-shot mutual exclusion algorithm $\mathcal{A}$ for n processes using $m+1$ shared variables, where the worst-case number of RMRs incurred by a process is at most $2t+2$.*

Proof Sketch: We design an algorithm $\mathcal{A}$ where processes correspond to the pebbles in $\mathcal{B}$ and spin variables correspond to the bins in $\mathcal{B}$. The algorithm appears in Figure 3 and is described informally as follows. Each process performs an atomic fetch&increment operation on TOKEN to acquire a unique name from $\{0, 1, \ldots, n-1\}$ (Line 1). Process i—the process that acquires name i—"simulates" pebble i. Pebble i visits the bins $b(i,1), b(i,2), \ldots, b(i, s(i))$ in that order, moving from $b(i,j)$ to $b(i, j+1)$ only when pebble $p(i,j)$ evaporates. Correspondingly, process i waits on $X[b(i,1)], X[b(i,2)], \ldots, X[b(i, s(i))]$ in that order, moving from $X[b(i,j)]$ to $X[b(i,j+1)]$ only after process $p(i,j)$ is enabled to enter the CS, which is indicated in the algorithm by the presence of $p(i,j)$ or a greater value in $X[b(i,j)]$. Thus, the termination of the last iteration of the for-loop indicates that process $p(i, s(i))$ is enabled to enter the CS; since $p(i, s(i)) = i$, process i enters the CS (Line 4). When process i leaves the CS, if it is not the last process, it enables process $i+1$ to enter the CS by writing $i+1$ in $X[b(i+1, s(i+1))]$ (Line 5). Our initialization of X[1] to 0 ensures that process 0 is enabled to enter the CS right at the start of the algorithm; thereafter, each process i is enabled to enter the CS by process $i-1$ in its exit at Line 5. Besides Mutual Exclusion and Starvation-Freedom, the algorithm satisfies FCFS [13]: processes enter the CS in the order they execute Line 1.

Process i incurs one RMR at Line 1, at most two RMRs at each execution of Line 3, and one RMR at Line 5, for a total of at most $2s(i)+2$ RMRs. Since t is the maximum of $s(i)$, over all i, the worst-case number of RMRs incurred by a process is at most $2t+2$. ■

We now extend the above ideas to design a general (i.e., multi-shot) mutual exclusion algorithm, where each process can repeatedly cycle through the Try, Critical, and Exit sections.

Theorem 4.2. *Suppose there is a bin-pebble strategy $\mathcal{B}$ for a set $\Pi = \{0, 1, \ldots, n-1\}$ of pebbles using m bins, where the maximum number of hits any pebble experiences is t. Then, there is a mutual exclusion algorithm $\mathcal{A}$ for n processes using $m+2$ shared variables, where the worst-case number of RMRs incurred by a process when executing the Try and Exit sections once is at most $2t+5$.*

Proof Sketch: The algorithm appears in Figure 4 and is described informally as follows. Processes acquire successive tokens by performing an atomic fetch&increment operation on TOKEN (Line 1). The algorithm divides processes into *batches* of size n and labels the batches *even* or *odd*, alternately. Thus, the first batch of processes acquiring tokens of $0, 1, \ldots, n-1$ is an even batch; the next batch of processes with tokens $n, n+1, \ldots, 2n-1$ is an odd batch; the next batch with tokens $2n, 2n+1, \ldots, 3n-1$ is an even batch; and so on. In Line 2, $t \bmod 2n$ is parsed into two fields: $parity \in \{0,1\}$ and $i \in \{0, 1, \ldots, n-1\}$. $parity$ is the high order bit of $t \bmod 2n$ (it is 0 if the process belongs to an even batch and 1 if it belongs to an odd batch); and i is the number formed by the $\lg n$ low order bits of $t \bmod 2n$, and represents the name that the process takes on during the execution of the Try and Exit sections. The single bit shared variable TOGGLE is used at Line 3 as a barrier: processes belonging to a new batch are prevented from crossing this barrier until *all* of the processes of the previous batch have completed the Exit section. Accordingly, a process waits at Line 3 until its parity tallies with the value in TOGGLE. Once past this barrier, a process acts similarly as in the earlier one-shot algorithm: Lines 4 to 7 are analogous to Lines 2 to 5 of the earlier algorithm, with just one difference: when waiting on a variable $X[b(i,j)]$, process i additionally checks if $X[b(i,j)]$'s parity bit agrees with its own (to prevent confusion from a value left in $X[b(i,j)]$ by the previous batch).

When process i leaves the CS, if it is not the last process in its batch, it writes in $X[b(i+1, s(i+1))]$ as before to enable process $i+1$ of its batch to enter the CS (Line 7). On the other hand, if process i is the last process in the current batch, it initializes $X[1]$ suitably—with the parity of the next batch and a value of 0 for the name field so that process 0 of the next batch will recognize that it is enabled—and then flips TOGGLE to release the next batch from the barrier (Lines 8 to 10). Besides Mutual Exclusion and Starvation-Freedom, the algorithm satisfies FCFS: processes enter the CS in the order they execute Line 1.

A process i incurs one RMR at Line 1, at most two RMRs at Line 3, at most two RMRs at each execution of Line 5, and at most two RMRs in the Exit section, for a total of at most $2s(i)+5$ RMRs. Since t is the maximum of $s(i)$, over all i, the worst-case number of RMRs incurred by a process in order to execute the Try and Exit sections once is at most $2t+5$. ■

Remark: The algorithm presented in Figure 4 uses an unbounded variable TOKEN. This can be easily fixed by adding the following line between Lines 1 and 2, incurring one more RMR but without affecting the correctness of the algorithm:

if $t = 2n-1$ **then** $t = \text{fetch\&add}(\text{TOKEN}, -2n)$

With this change, TOKEN becomes a bounded variable that only takes on values from $\{0, 1, \ldots, 3n-1\}$.

5. BIN PEBBLE STRATEGIES

In this section, we present two bin-pebble strategies for n pebbles using m bins, one for the case of $m \le \log n$ and another for $m \ge \log^{1+\epsilon} n$, where $\epsilon > 0$ is a real constant.

THEOREM 5.1. *There exists a bin-pebble strategy for n pebbles using $m \le \log n$ bins, where the maximum number of hits that a pebble experiences is $O\left(mn^{1/m}\right)$.*

PROOF. The strategy is defined in Figure 5. The sequence of n steps, as defined, is a valid strategy: in each step, a non-empty bin is shaken; one pebble evaporates; and some pebbles may be transferred from the shaken bin to other bins.

It follows from the definition and the invariant in Figure 5 that any non-empty bin is shaken at most $\lceil n^{1/m} \rceil$ times before it becomes empty, and a pebble is never transferred to a lower bin. Hence, any pebble is at most $m \cdot \lceil n^{1/m} \rceil$ times in a bin that is shaken. ■

We now present a bin-pebble strategy for large values of m.

Let the bins be numbered $1, 2, \ldots, m$ and let $np(i,k)$ denote the number of pebbles in bin k after the ith step. We ensure the following key invariant for all $0 \le i \le n$:
Invariant SMALL-M-INV(i): $np(i,k) \le \lceil n^{(m-k+1)/m} \rceil$.

1. Initially, all n pebbles are in bin 1. Therefore, SMALL-M-INV(0) holds.
2. **for** $i = 1$ **to** n
3. Let k be the highest non-empty bin.
   Shake bin k:
   (a) Remove one pebble from bin k.
   (b) If $k < m$, transfer $\min\{\lceil n^{(m-k)/m} \rceil, np(i-1,k)-1\}$ pebbles to bin $k+1$ from bin k.
   Since $np(i-1,k+1)=0$ and at most $\lceil n^{(m-k)/m} \rceil$ pebbles are transferred to bin $k+1$, SMALL-M-INV(i) holds. The invariant trivially holds if no pebbles are transferred (i.e., $k=m$).

Figure 5: A bin-pebble strategy for n pebbles using m bins when $m \le \log n$.

Let r and d be the integers such that $rd < m < (r+1)d$ and $d = \lceil \frac{\log n}{\log r} \rceil$. It is easy to verify that $\log r = \Omega(\log m)$. Initially, all n pebbles are in bin 1. Pick any rd bins from the $m-1$ empty bins and divide them into d groups: r bins in each group. Let the groups be numbered $0, 1, \ldots, d-1$. Let $np(i,k)$ denote the number of pebbles in bin k after the ith step. We ensure the following key invariant for all $1 \le i \le n$:
Invariant LARGE-M-INV(i): For any bin k in group l, $np(i,k) \le r^l$.

1. Shake bin 1: Remove one pebble and transfer the remaining $n-1$ pebbles to the r bins in group $d-1$.
   Since $n \le r^d$, we can transfer the pebbles so that no bin in group $d-1$ gets more than r^{d-1} pebbles and LARGE-M-INV(1) holds.
2. **for** $i = 2$ **to** n
3. Let l be the smallest group with a non-empty bin and let k be a non-empty bin in group l.
   Shake bin k:
   (a) Remove one pebble from the bin k.
   (b) If $l > 0$, transfer $\min\{r^l, np(i-1,k)-1\}$ pebbles to the bins in group $l-1$.
   This transfer can be done in a way that LARGE-M-INV(i) holds because all bins in group $l-1$ are empty and there are r bins in the group. Hence, they can absorb up to r^l pebbles.

Figure 6: A bin-pebble strategy for n pebbles using m bins when $m \ge \log^{1+\epsilon} n$.

Theorem 5.2. *There exists a bin-pebble strategy for n pebbles using $m \geq \log^{1+\epsilon} n$ bins (for some constant $\epsilon > 0$), where the maximum number of hits that a pebble experiences is $O\left(\frac{\log n}{\log m}\right)$.*

Proof. The strategy is defined in Figure 6 and is clearly a valid strategy. It follows from the definition and the invariant in Figure 6 that any non-empty bin is shaken at most once before it becomes empty, and a pebble is never transferred to a bin of the same or higher group after the first shake. Hence, any pebble is at most $d + 1$ times in a bin that is shaken. Thus, the maximum number of hits that a pebble experiences is at most $d+1 = \lceil \frac{\log n}{\log r} \rceil + 1 = O\left(\frac{\log n}{\log m}\right)$. ■

6. ACKNOWLEDGMENTS

We thank the reviewers for their helpful feedback.

7. REFERENCES

[1] Anderson, T. E. The performance of spin lock alternatives for shared-memory multiprocessors. *IEEE Trans. Parallel Distrib. Syst. 1*, 1 (1990), 6–16.

[2] Attiya, H., Hendler, D., and Woelfel, P. Tight RMR lower bounds for mutual exclusion and other problems. In *STOC '08: Proceedings of the 40th annual ACM symposium on Theory of computing* (New York, NY, USA, 2008), ACM, pp. 217–226.

[3] Bender, M. A., and Gilbert, S. Mutual exclusion with $O(\log^2 \log n)$ amortized work. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science* (2011), pp. 728–737.

[4] Burns, J. E., and Lynch, N. A. Bounds on shared memory for mutual exclusion. *Inf. Comput. 107* (December 1993), 171–184.

[5] Craig, T. Queuing spin lock algorithms to support timing predictability. In *Proceedings of the 14th IEEE Real-time Systems Symposium* (1993), IEEE, pp. 148–156.

[6] Cypher, R. The communication requirements of mutual exclusion. In *SPAA '95: Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures* (New York, NY, USA, 1995), ACM, pp. 147–156.

[7] Dijkstra, E. W. Solution of a problem in concurrent programming control. *Commun. ACM 8*, 9 (1965), 569.

[8] Fan, R., and Lynch, N. An $\Omega(n \log n)$ lower bound on the cost of mutual exclusion. In *PODC '06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing* (New York, NY, USA, 2006), ACM, pp. 275–284.

[9] Fich, F. E., Hendler, D., and Shavit, N. Linear lower bounds on real-world implementations of concurrent objects. In *FOCS '05: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 165–173.

[10] Granunke, G., and Thakkar, S. Synchronization algorithms for shared-memory multiprocessors. *Computer 23*, 6 (1990), 60–69.

[11] Hendler, D., and Woelfel, P. Randomized mutual exclusion with sub-logarithmic RMR-complexity. *Distributed Computing 24* (2011), 3–19.

[12] Knuth, D. E. Additional comments on a problem in concurrent programming control. *Commun. ACM 9*, 5 (1966), 321–322.

[13] Lamport, L. A new solution of Dijkstra's concurrent programming problem. *Commun. ACM 17*, 8 (1974), 453–455.

[14] Lamport, L. The mutual exclusion problem: part ii - statement and solutions. *J. ACM 33*, 2 (1986), 327–348.

[15] Mellor-Crummey, J. M., and Scott, M. L. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Trans. Comput. Syst. 9*, 1 (1991), 21–65.

[16] Raynal, M., and Beeson, D. *Algorithms for mutual exclusion.* MIT Press, Cambridge, MA, USA, 1986.

[17] Yang, J.-H., and Anderson, J. H. A fast, scalable mutual exclusion algorithm. *Distributed Computing 9* (1994), 9–1.